PHẦN 1. MỘT SỐ BÀI TOÁN GIẢI QUYẾT BẰNG CHIẾN LƯỢC CHIA ĐỂ TRỊ

I. Bài toán tìm kiếm nhị phân

1. **Mô tả bài toán**: Bài toán tìm kiếm gồm dữ liệu đầu vào là một mảng n phần tử đã có thứ tự, một khóa *key* kèm theo để so sánh giữa các phần tử trong mảng. Kết quả của bài toán là có phần tử nào trong mảng bằng với *key* không ?

   Xây dựng thuật giải tìm kiếm nhị phân từ thuật giải chia để trị tổng quát.

2. **Ý tưởng** : Giả sử ta có mảng A có thứ tự tăng dần. Khi đó $A_i < A_j$ với i<j.

- **Divide**: xác định phần tử giữa mảng là A_{mid}. Nếu phần tử cần tìm bằng phần tử này thì trả về vị trí tìm được và kết thúc. Nếu phần tử cần tìm nhỏ hơn A_{mid} thì ta chỉ cần tìm trong dãy con bên trái A_{mid}. Nếu phần tử cần tìm lớn hơn phần tử này thì ta chỉ cần tìm trong dãy con bên phải A_{mid}.
- **Conquer**: Tiếp tục tìm kiếm trong các dãy con đến khi tìm thấy khóa key hoặc đến khi hết dãy khi không có key trong dãy.
- **Combine**: Không cần trong trường hợp này.

3. **Mã giả giải thuật**

```
Binary_Search( A, n, key)
{
    left = 0; // vị trí phần tử đầu tiên trong mảng
    right = n-1; // vị trí phần tử cuối cùng trong mảng
    while (left <= right)
    {
        mid = (left + right)/2;        //vị trí giữa mảng
        if (A[mid] == key)
                return mid;
        if (key < A[mid])
          right = mid – 1 ;
        else
          left = mid + 1;
    }
    return -1; // không tìm thấy key trong mảng nên trả về vị trí -1.
}
```

II. Bài toán sắp xếp theo giải thuật Merge Sort

1. **Mô tả bài toán**: Bài toán sắp xếp gồm dữ liệu đầu vào là một mảng các phần tử. Kết quả của bài toán là mảng đã được xếp theo thứ tự (tăng hoặc giảm).

Xây dựng thuật giải sắp xếp Merge Sort từ thuật giải chia để trị tổng quát.

2. **Ý tưởng** : Giả sử ta có mảng A có n phần tử.

- **Divide**: Chia dãy n phần tử cần sắp xếp thành 2 dãy con, mỗi dãy có n/2 phần tử.
- **Conquer**: Sắp xếp các dãy con bằng cách gọi đệ quy Merge Sort. Dãy con chỉ có 1 phần tử thì mặc nhiên có thứ tự, không cần sắp xếp.
- **Combine**: Trộn 2 dãy con đã sắp xếp để tạo thành dãy ban đầu có thứ tự.

3. **Mã giả giải thuật**

```
Merge_Sort (A, 0, n)
{
        left = 0; // vị trí phần tử đầu tiên của dãy.
        right = n-1; // ví trị phần tử cuối cùng của dãy
        if (left < right)
        {
         mid = (left + right)/2;     //vị trí phần tử ở giữa dãy
         Merge_Sort (A, left, mid);        // Gọi hàm Merge_Sort cho nửa dãy con đầu
         Merge_Sort (A, mid+1, right); // Gọi hàm Merge_Sort cho nửa dãy con cuối
         Merge (A, left, mid, right);        //Hàm trộn 2 dãy con có thứ tự thành dãy ban đầu có thứ
tự
        }
}
```

//----------

```
Merge (A, left, mid, right)
{
    n1 = mid - left + 1; //độ dài nửa dãy đầu của A.
    n2 = right - mid; // độ dài nửa dãy sau của A
    L[], R[];        //L là dãy chứa nửa dãy đầu của A; R là dãy chứa nửa dãy sau của A.
    for i = 0  to n1-1
            L[i] = A[left+i];       //chép nửa dãy đầu của A vào L.
    for j = 0 to n2-1
            R[j] = A[mid + j + 1]          //chép nửa dãy sau của A vào R.
    i=0
    j=0
    for k = left to right             // L và R lại vào A sao cho A có thứ tự tăng dần.
```

```
        if (L[i] <= R[j])
        {
                A[k] = L[i]
                i = i+1
        }
        else
        {
                A[k] = R[j]
                j = j + 1
        }
}
```

III. Bài toán nhân 2 số nguyên lớn

1. **Mô tả bài toá**n: Trong các ngôn ngữ lập trình đều có kiểu dữ liệu số nguyên (chẳng hạn kiểu integer trong Pascal, Int trong C…), nhưng nhìn chung các kiểu này đều có miền giá trị hạn chế (chẳng hạn từ -32768 đến 32767) nên khi có một ứng dụng trên số nguyên lớn (hàng chục, hàng trăm chữ số) thì kiểu số nguyên định sẵn không đáp ứng được. Trong trường hợp đó, người lập trình phải tìm một cấu trúc dữ liệu thích hợp để biểu diễn cho một số nguyên, chẳng hạn ta có thể dùng một chuỗi kí tự để biểu diễn cho một số nguyên, trong đó mỗi kí tự lưu trữ một chữ số. Để thao tác được trên các số nguyên được biểu diễn bởi một cấu trúc mới, người lập trình phải xây dựng các phép toán cho số nguyên như phép cộng, phép trừ, phép nhân… Bài toán sau đây sẽ đề cập đến bài toán nhân hai số nguyên lớn.

Xét bài toán nhân 2 số nguyên lớn X và Y, mỗi số có n chữ số.

Theo cách nhân thông thường mà ta đã được học ở phổ thông thì phép nhân được thực hiện như sau: nhân từng chữ số của Y với X (kết quả được dịch trái 1 vị trí sau mỗi lần nhân) sau đó cộng các kết quả lại.

Ví dụ: X = 2357, Y = 4891, ta đặt tính nhân như hình bên

```
      2357
       ×
       4891
     ----------
      2357
    + 21213
     18856
     9428
-------------------------
   11528087
```

Việc nhân từng chữ số của X và Y tốn n^2 phép nhân (vì X và Y có n chữ số). Nếu phép nhân một chữ số của X cho một chữ số của Y tốn O(1) thời gian, thì độ phức tạp giải thuật của giải thuật nhân X và Y này là $O(n^2)$.

Xây dựng thuật giải nhân số nguyên lớn từ thuật giải chia để trị tổng quát.

2. **Ý tưởng** :

Áp dụng kĩ thuật "chia để trị" vào phép nhân các số nguyên lớn, ta chia mỗi số nguyên lớn X và Y thành các số nguyên lớn có n/2 chữ số. Để việc phân tích giải

thuật đơn giản, ta giả sử n là luỹ thừa của 2, còn về khía cạnh lập trình, vì máy xử lý nên ta vẫn có thể viết chương trình với n bất kì.

- Biểu diễn X và Y dưới dạng sau: X = A.10n/2 + B ; Y = C.10n/2 + D.
- Trong đó A, B, C, D là các số có n/2 chữ số. Chẳng hạn với X = 7853 thì A = 78 và B=53 vì $78*10^2 + 53 = 7853 = X$
- Do đó, với cách biểu diễn trên thì $XY = A.C.10^n + (A.D + B.C)10^{n/2} + B.D$ (*)

Khi đó thay vì nhân trực tiếp 2 số có n chữ số, ta phân tích bài toán ban đầu thành một số bài toán nhân 2 số có n/2 chữ số. Sau đó, ta kết hợp các kết quả trung gian theo cách phân tích và công thức (*). Việc phân chia này sẽ dẫn đến các bài toán nhân 2 số có 1 chữ số. Đây là bài toán cơ sở. Tóm lại các bước giải thuật chia để trị cho bài toán trên như sau:

- **Divide:** Chia bài toán nhân 2 số nguyên lớn X, Y có n chữ số thành các bài toán nhân các số có n/2 chữ số.
- **Conquer**: tiếp tục chia các bài toán nhân sao cho đưa về các bài toán nhân 2 số có 1 chữ số.
- **Combine**: Tổng hợp các kết quả trung gian theo công thức (*).

3. **Mã giả giải thuật** : nhân 2 số nguyên lớn X, Y có n chữ số.

```
Big_Number_Multi (Big_Int X, Big_Int Y, int n)
{
        Big_Int m1, m2, m3, A, B, C, D;
        int s; //lưu dấu của tích XY
        s = sign(X) * sign(Y); //sign(X) trả về 1 nếu X dương; -1 là âm; 0 là X = 0

        X = abs(X);
        Y = abs(Y);
        if (n==1) // X, Y có 1 chữ số
                return X*Y*s;
        else
        {
                        A = left (X, n/2); // số có n/2 chữ số đầu của X.
                        B = right(X, n/2); // số có n/2 chữ số cuối của X.
                        C = left(Y, n/2);// số có n/2 chữ số đầu của Y
                        D = right (Y, n/2);// số có n/2 chữ số cuối của Y
```

```
                m1 = Big_Number_Multi (A, C, n/2);
                m2 = Big_Number_Multi (A-B, D-C, n/2);
                m3 = Big_Number_Multi (B, D, n/2);
                return s* (m1*10^n + (m1 + m2 + m3)*10^(n/2) +m3);
        }
}
```

PHẦN 2. MỘT SỐ BÀI TOÁN GIẢI QUYẾT BẰNG CHIẾN LƯỢC THAM LAM

I. Bài toán cái ba lô

1. Mô tả bài toán :

Cho một cái ba lô có thể đựng một trọng lượng W và n loại đồ vật, mỗi đồ vật i có một trọng lượng g_i và một giá trị v_i. Tất cả các loại đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá W và tổng giá trị là lớn nhất.

2. Ý tưởng tham lam giải bài toán

Theo yêu cầu của bài toán thì ta cần những đồ vật có giá trị cao mà trọng lượng lại nhỏ để sao cho có thể mang được nhiều đồ có giá trị nhất, sẽ là hợp lý khi ta quan tâm đến yếu tố "đơn giá" của từng loại đồ vật tức là tỷ lệ giá trị/trọng lượng. Đơn giá càng cao thì đồ càng quý. Từ đó ta có kĩ thuật tham ăn để áp dụng cho bài toán này là:

- Bước 1 : Tính đơn giá cho các loại đồ vật.
- Bước 2 : Xét các loại đồ vật theo thứ tự đơn giá từ lớn đến nhỏ.
- Bước 3 : Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà trọng lượng còn lại ba lô cho phép.
- Bước 4 : Xác định trọng lượng còn lại của ba lô và quay lại bước 3 cho đến khi không còn có thể chọn được đồ vật nào nữa.

Ví dụ :

Cho ba lô có trọng lượng là 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng sau đây:

Loại đồ vật	Trọng lượng	Giá trị
A	15	30
B	10	25
C	2	2
D	4	6

Từ bảng đã cho ta tính đơn giá cho các loại đồ vật và sắp xếp các loại đồ vật này theo thứ tự đơn giá giảm dần, ta có bảng sau:

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
B	10	25	2.5
A	15	30	2
D	4	6	1.5
C	2	2	1

Theo đó thì thứ tự ưu tiên để chọn đồ vật là B, A, D và cuối cùng là C.

Vật B được xét đầu tiên và ta chọn tối đa 3 cái vì mỗi cái vì trọng lượng mỗi cái là 10 và ba lô có trọng lượng 37. Sau khi đã chọn 3 vât loại B, trọng lượng còn lại trong ba lô là 37- 3*10 =7. Ta xét đến vật A, vì A có trọng lượng 15 mà trọng lượng còn lại của balô chỉ còn 7 nên không thể chọn vật A. Xét vật D và ta thấy có thể chọn 1 vật D, khi đó trọng lượng còn lại của ba lô là 7-4 = 3. Cuối cùng ta chọn được một vật C.

Như vậy chúng ta đã chọn 3 cái loại B, một cái loại D và 1 cái loại C.

Tổng trọng lượng = 3*10 + 1*4 + 1*2 = 36 và tổng giá trị = 3*25+1*6+1*2 = 83.

3. Mã giả giải bài toán cái ba lô bằng kĩ thuật tham lam

Tổ chức dữ liệu:

- Mỗi đồ vật được biểu diễn bởi một cấu trúc chứa các thông tin:
  - + Ten: Lưu trữ tên đồ vật.
  - + Trong_luong: Lưu trữ trọng lượng của đồ vật.
  - + Gia_tri: Lưu trữ giá trị của đồ vật
  - + Don_gia: Lưu trữ đơn giá của đồ vật
  - + Phuong_an: Lưu trữ số lượng đồ vật được chọn theo phương án.
- Danh sách các đồ vật được biểu diễn bởi một mảng các đồ vật.

```
const int MAX_SIZE = 100;
struct Do_vat
{
    char Ten [20];
    float Trong_luong, Gia_tri, Don_gia;
    int Phuong_an;
};
 Do_vat dsdv[MAX_SIZE];
```

```
void Greedy (Do_vat  dsdv[], float W)
{
    /*Sắp xếp mảng dsdv theo thứ tự giảm của don_gia*/
    for (int i = 0; i < n; i++) {
            dsdv[i].Phuong_an = chon(dsdv[i].Trong_luong, W);
            W = W – dsdv[i].Phuong_an * dsdv[i].Trong_luong;
    }
}
```

Trong đó hàm *chon(trong_luong, W)* nhận vào trọng lượng trong_luong của một vật và trọng lượng còn lại W của ba lô, trả về số lượng đồ vật được chọn, sao cho tổng trọng lượng của các vật được chọn không lớn hơn W. Nói riêng, trong trường hợp trong_luong và W là hai sô nguyên thì chon(Trong_luong, W) chính là W / Trong_luong.

4. Một số biến thể của bài toán cái ba lô như sau:

a. Mỗi đồ vật i chỉ có một số lượng s_i. Với bài toán này khi lựa chọn vật i ta không được lấy một số lượng vượt quá s_i.

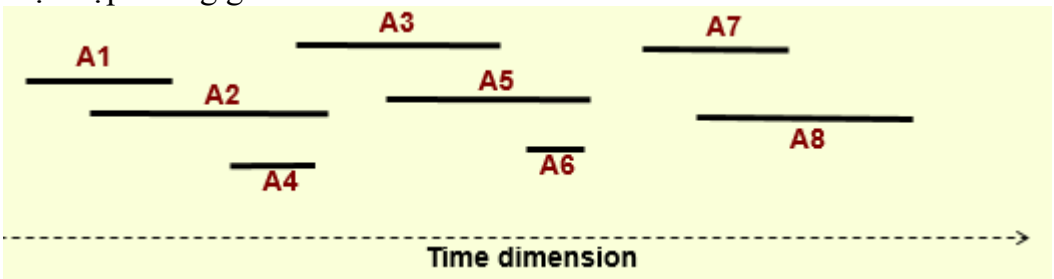
b. Mỗi đồ vật chỉ có một cái. Với bài toán này thì với mỗi đồ vật ta chỉ có thể chọn hoặc không chọn.

II. Bài toán Lựa chọn hoạt động - Activity-Selection Problem

1. Mô tả bài toán

- Cho tập các hoạt động $A_1, A_2, \ldots A_n$ (ví dụ : các cuộc họp, người thuyết trình,…)
- Mỗi hoạt động có thời gian bắt đầu (start time - S_i) và thời gian kết thúc (end time – E_i).
- Các hoạt động sẽ sử dụng chung nguồn tài nguyên (ví dụ : phòng họp, …).
- Mục tiêu của bài toán : tối đa hóa số lượng các hoạt động sử dụng nguồn tài nguyên, với điều kiện không có hai hoạt động bị trùng lắp.

Ví dụ : Có n cuộc họp và một phòng họp, mỗi cuộc họp có thời gian bắt đầu và kết thúc. Cần xếp lịch các cuộc họp vào phòng họp sao cho được số cuộc họp nhiều nhất có thể, nhưng không được có 2 cuộc họp chồng giờ lên nhau.



Các kết quả có thể có để xếp các cuộc họp A_i vào 1 phòng họp là :
{A1, A3, A8}
{A1, A4, A5, A7}
{A2, A5, A8}
….

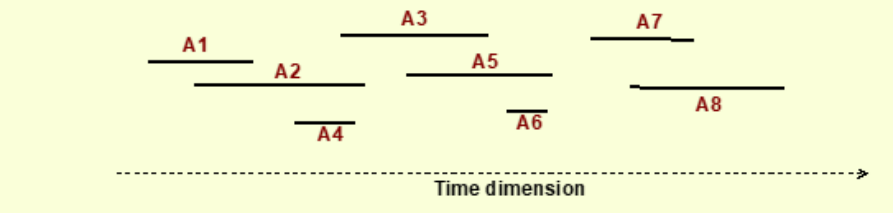
2. Ý tưởng thuật toán tham lam giải quyết bài toán

Bước 1 : Chọn hoạt động có thời gian kết thúc trước nhất (thời gian kết thúc nhỏ nhất) → trực giác : để lại khoảng thời gian lớn nhất có thể cho nhiều hoạt động được chọn.

Bước 2 : Xóa các hoạt động không phù hợp. Nghĩa là các hoạt động có thời gian trùng lắp với hoạt động vừa chọn ở bước 1. → bởi vì các hoạt động này sẽ không được chọn.

Bước 3. Lặp lại bước 1, 2 cho đến khi hết các hoạt động.

Ví dụ :



A:	i	1	2	3	4	5	6	7	8
	S_i	1	2	7	5	10	16	21	23
	E_i	4	9	15	8	18	17	27	30

Chọn A1, xóa A2, chọn A4, xóa A3, chọn A6, chọn A7, xóa A8. → {A1, A4, A6, A7}

III. Bài toán đi đường của người giao hàng.

1. Mô tả bài bài toán

Một trong những bài toán nổi tiếng áp dụng kỹ thuật tham lam để tìm cách giải quyết đó là bài toán tìm đường đi của người giao hàng (TSP - Traveling Salesman Problem). Bài toán được mô tả như sau: *Có một người giao hàng cần đi giao hàng tại n thành phố. Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu.*

Yêu cầu :

+ Mỗi thành phố chỉ đến một lần.
+ Khoảng cách từ một thành phố đến các thành phố khác là xác định được.
+ Giả thiết rằng mỗi thành phố đều có đường đi đến các thành phố còn lại.
+ Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, có thể là cước phí di chuyển hoặc thời gian di chuyển. Ta gọi chung là độ dài.

Hãy tìm một chu trình (một đường đi khép kín thỏa mãn các điều kiện trên) sao cho tổng độ dài chu trình là nhỏ nhất.

2. Ý tưởng tham lam giải quyết bài toán

Bài toán này cũng được gọi là bài toán người du lịch. Một cách tổng quát, có thể không tồn tại một đường đi giữa hai thành phố a và b nào đó. Trong trường hợp đó ta cho một đường đi ảo giữa a và b với độ dài bằng ∞. Bài toán có thể biểu diễn bởi một đồ thị vô hướng có trọng số G=(V, E), trong đó mỗi thành phố được biểu diễn bởi một đỉnh, cạnh nối hai đỉnh biểu diễn cho đường đi giữa hai thành phố và trọng số của cạnh là khoảng cách giữa hai thành phố. Một chu trình đi qua tất cả các đỉnh của G, mỗi đỉnh một lần duy nhất, được gọi là chu trình Hamilton. Vấn đề là tìm một chu trình Hamilton mà tổng độ dài các cạnh là nhỏ nhất.

Dễ dàng thấy rằng, với phương pháp vét cạn ta xét tất cả các chu trình, mỗi chu trình tính tổng độ dài các cạnh của nó rồi chọn một chu trình có tổng độ dài nhỏ nhất. Tuy nhiên chúng ta phải xét tất cả $\frac{(n-1)!}{2}$ chu trình. Thực vậy, do mỗi chu trình đều đi qua tất cả các đỉnh (thành phố) nên ta có thể cố định một đỉnh. Từ đỉnh này ta có n-1 cạnh tới n-1 đỉnh khác, nên ta có n-1 cách chọn cạnh đầu tiên của chu trình. Sau khi đã chọn được cạnh đầu tiên, chúng ta còn n-2 cách chọn cạnh thứ hai, do đó ta có (n-1)(n-2) cách chọn hai cạnh. Cứ lý luận như vậy ta sẽ thấy có (n-1)! cách chọn một chu trình. Tuy nhiên với mỗi chu trình ta chỉ quan tâm đến tổng độ dài các cạnh chứ không quan tâm đến hướng đi theo chiều dương hay âm vì vậy có tất cả $\frac{(n-1)!}{2}$ phương án. Đó là một giải thuật có độ phức tạp là một thời gian mũ. Vì vậy khi áp dụng kỹ thuật tham ăn ở một số bài toán chúng ta chỉ có thể thu được các giải quyết tốt chứ không thể là tối ưu nhất.

Áp dụng kỹ thuật tham ăn vào bài toán này như sau :

Bước 1 : Sắp xếp các cạnh theo thứ tự tăng của độ dài.

Bước 2 : Xét các cạnh có độ dài từ nhỏ đến lớn để đưa vào chu trình.

Bước 3 : Một cạnh sẽ được đưa vào chu trình nếu cạnh đó thỏa hai điều kiện sau:

+ *Không tạo thành một chu trình thiếu (không đi qua đủ n đỉnh)*
+ *Không tạo thành một đỉnh có cấp ≥ 3 (tức là không được có nhiều hơn hai cạnh xuất phát từ một đỉnh, do yêu cầu của bài toán là mỗi thành phố chỉ được đến một lần: một lần đến và một lần đi).*

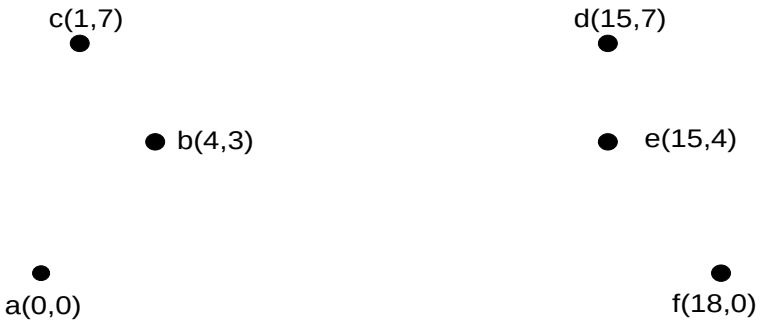
Bước 4 : Lặp lại bước 3 cho đến khi xây dựng được một chu trình.

Với kĩ thuật này ta chỉ cần n(n-1)/2 phép chọn nên ta có một giải thuật cần $O(n^2)$ thời gian.

Ví dụ 1:

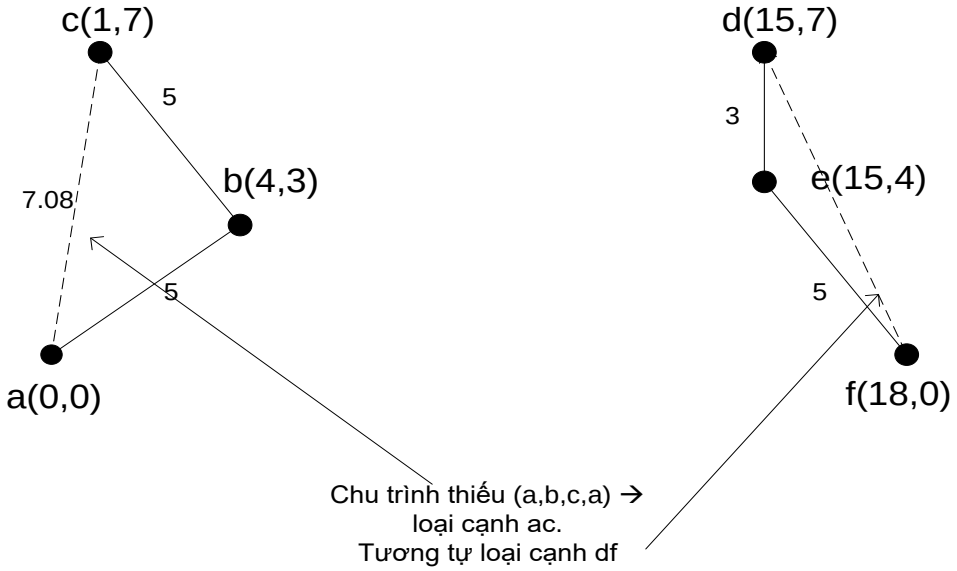
Cho bài toán TSP với 6 điểm có tọa độ tương ứng :

a(0,0), b(4,3), c(1,7), d(15,7), e(15,4) và f(18,0).

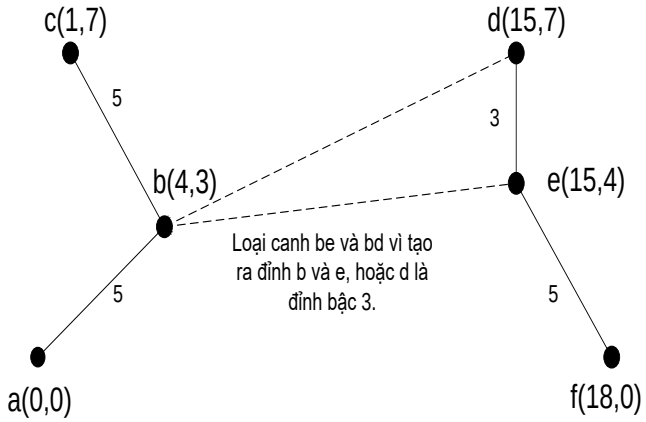


Do có 6 đỉnh nên có tất cả 15 cạnh. Đó là các cạnh: ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df và ef. Độ dài các cạnh ở đây là khoảng cách Euclide (khoảng cách 2 điểm A, B = $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$).

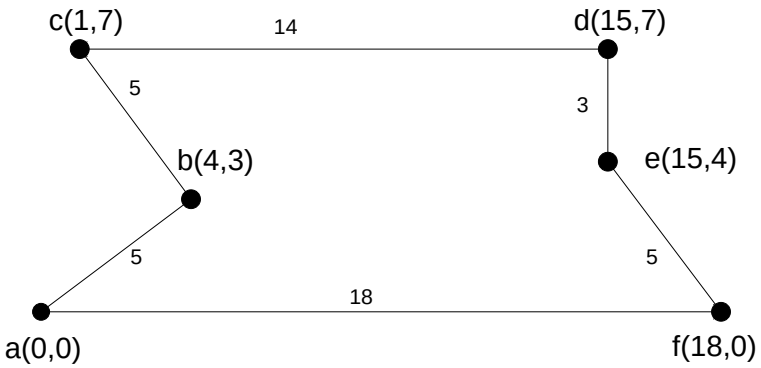
stt	Cạnh	X_1	Y_1	X_2	Y_2	Độ dài
1	de	15	7	15	4	3
2	ab	0	0	4	3	5
3	bc	4	3	1	7	5
4	ef	15	4	18	0	5
5	ac	0	0	1	7	7.08
6	df	15	7	18	0	7.62
7	be	4	3	15	4	11.05
8	bd	4	3	15	7	11.7
9	cd	1	7	15	7	14
10	bf	4	3	18	0	14.32
11	ce	1	7	15	4	14.32
12	ae	0	0	15	4	15.52
13	ad	0	0	15	7	16.55
14	af	0	0	18	0	18
15	cf	1	7	18	0	18.38



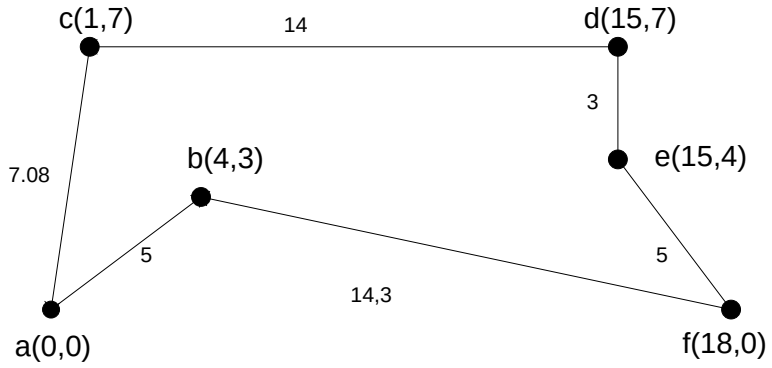
Trong số các cạnh thì canh de = 3 là nhỏ nhất, nên de được chọn vào chu trình. Kế đến là 3 cạnh ab, bc và ef đều có độ dài là 5. Cả 3 cạnh đều thỏa mãn hai điều kiện nói trên, nên đều được chọn vào chu trình. Cạnh có độ dài nhỏ kế tiếp là ac = 7.08, nhưng không thể đưa cạnh này vào chu trình vì nó sẽ tạo ra chu trình thiếu (a-b-c-a). Cạnh df cũng bị loại vì lý do tương tự. Cạnh be được xem xét nhưng rồi cũng bị loại do tạo ra đỉnh b và đỉnh e có cấp 3. Tương tự chúng ta cũng loại bd.



Cạnh cd là cạnh tiếp theo được xét và được chọn. Cuối cùng ta có chu trình a-b-c-d-e-f-a với tổng độ dài là 50. Đây chỉ là một phương án tốt nhưng chưa tối ưu.



Dựa vào giả thiết bài toán trên chúng ta có kết quả tối ưu của bài toán là chu trình a-c-d-e-f-b-a với tổng độ dài là 48.38.



3. **Mã giả của giải thuật dành cho bài toán tìm đường đi của người giao hàng**.

```
void TSP()
{
    /*E là tập hợp các cạnh, Chu_trinh là tập hợp các cạnh được chọn để đưa vào chu trình, mở đầu Chu_trinh rỗng*/
    /*Sắp xếp các cạnh trong E theo thứ tự tăng của độ dài*/
    Chu_Trinh = Φ;
    Gia = 0.0;
    while (E != Φ) {
        if (cạnh e có thể chọn)
        {
            Chu_Trinh = Chu_Trinh + [e];
            Gia = Gia + độ dài của e;
        }
        E= E-[e];
    }
}
```

4. **Một cách tiếp cận khác của kĩ thuật tham lam vào bài toán này là**:

- Bước 1. Xuất phát từ một đỉnh bất kỳ, chọn một cạnh có độ dài nhỏ nhất trong tất cả các cạnh đi ra từ đỉnh đó để đến đỉnh kế tiếp.
- Bước 2. Từ đỉnh kế tiếp ta lại chọn một cạnh có độ dài nhỏ nhất đi ra từ đỉnh này thoả mãn hai điều kiện nói trên để đi đến đỉnh kế tiếp.
- Bước 3. Lặp lại bước 2 cho đến khi đi tới đỉnh n thì quay trở về đỉnh xuất phát.

IV. Bài toán Nén dữ liệu mã hóa Huffman

1. **Mô tả bài toá**n

Mã hóa Huffman (David A. Huffman) là một thuật toán mã hóa dùng để nén dữ liệu. Dựa trên bảng tần suất xuất hiện các kí tự cần mã hóa để xây dựng một bộ mã nhị phân cho các kí tự đó sao cho dung lượng (số bit) sau khi mã hóa là nhỏ nhất.

2. **Ý tưởng thuật toá**n

2.1. **Mã tiền tố (prefix-free binary code**)

Để mã hóa các kí hiệu (kí tự, chữ số, ...) ta thay chúng bằng các xâu nhị phân, được gọi là từ mã của kí hiệu đó. Chẳng hạn bộ mã ASCII, mã hóa cho 256 kí hiệu là biểu diễn nhị phân của các số từ 0 đến 255, mỗi từ mã gồm 8 bít. Trong ASCII từ mã của kí tự "a" là 1100001, của kí tự "A" là 1000001. Trong cách mã hóa này các từ mã của tất cả 256 kí hiệu có độ dài bằng nhau (mỗi từ mã 8 bít). Nó được gọi là mã hóa với độ dài không đổi.

Khi mã hóa một tài liệu có thể không sử dụng đến tất cả 256 kí hiệu. Hơn nữa trong tài liệu chữ cái "a" chỉ có thể xuất hiện 1000000 lần còn chữ cái "A" có thể chỉ xuất hiện 2, 3 lần. Như vậy ta có thể không cần dùng đủ 8 bít để mã hóa cho một ký hiệu, hơn nữa độ dài (số bít) dành cho mỗi kí hiệu có thể khác nhau, kí hiệu nào xuất hiện nhiều lần thì nên dùng số bít ít, ký hiệu nào xuất hiện ít thì có thể mã hóa bằng từ mã dài hơn. Như vậy ta có việc mã hóa với độ dài thay đổi. Tuy nhiên, nếu mã hóa với độ dài thay đổi, khi giải mã ta làm thế nào phân biệt được xâu bít nào là mã hóa của ký hiệu nào. Một trong các giải pháp là dùng các dấu phẩy (",") hoặc một kí hiệu quy ước nào đó để tách từ mã của các kí tự đứng cạnh nhau. Nhưng như thế số các dấu phẩy sẽ chiếm một không gian đáng kể trong bảng mã. Một cách giải quyết khác dẫn đến khái niệm mã tiền tố.

Mã tiền tố là bộ các từ mã của một tập hợp các kí hiệu sao cho từ mã của mỗi ký hiệu không là tiền tố (phần đầu) của từ mã một ký hiệu khác trong bộ mã ấy. Đương nhiên mã hóa với độ dài không đổi là mã tiền tố.

Ví dụ:

Giả sử mã hóa từ "ARRAY", tập các ký hiệu cần mã hóa gồm 3 chữ cái "A","R","Y".

- Nếu mã hóa bằng các từ mã có độ dài bằng nhau ta dùng ít nhất 2 bit cho một chữ cái chẳng hạn "A"=00, "R"=01, "Y"=10. Khi đó mã hóa của cả từ là 0001010010. Để giải mã ta đọc hai bit một và đối chiếu với bảng mã.
- Nếu mã hóa "A"=0, "R"=01, "Y"=11 thì bộ từ mã này không là mã tiền tố vì từ mã của "A" là tiền tố của từ mã của "R". Để mã hóa cả từ ARRAY phải đặt dấu ngăn cách vào giữa các từ mã 0,01,01,0,11
- Nếu mã hóa "A"=0, "R"=10, "Y"=11 thì bộ mã này là mã tiền tố. Với bộ mã tiền tố này khi mã hóa xâu "ARRAY" ta có 01010011.

2.2. **Cây Huffma**n

- Là cây nhị phân, mỗi nút chứa ký tự và trọng số (tần suất của ký tự đó).
- Mỗi ký tự được biểu diễn bằng 1 nút lá (tính tiền tố)
- Nút cha có tổng ký tự, tổng trọng số của 2 nút con.
- Các nút có trọng số, ký tự tăng dần từ trái sang phải.

- Các nút có trọng số lớn nằm gần nút gốc.
- Các nút có trọng số nhỏ nằm xa nút gốc hơn.

2.3. **Mã Huffman**

- Là chuỗi nhị phân được sinh ra dựa trên cây Huffman.
- Mã Huffman của ký tự là đường dẫn từ nút gốc đến nút lá đó: sang trái ta được bit 0, sang phải ta được bit 1
- Có độ dài biến đổi (tối ưu bảng mã).
- Các ký tự có tần suất lớn có độ dài ngắn.
- Các ký tự có tần suất nhỏ có độ dài dài hơn.

Mỗi mã phi tiền tố có thể biểu diễn bởi một cây nhị phân T mà mỗi lá của nó tương ứng với một chữ cái và cạnh của nó được gán cho một trong hai số 0 hoặc 1. Mã của một chữ cái *c* là một dãy nhị phân gồm các số gán cho các cạnh trên đường đi từ gốc đến lá tương ứng với *c*.

Bài toán: Tìm cách mã hóa tối ưu, tức là tìm cây nhị phân T làm tối thiểu hóa tổng độ dài có trọng số. $B(T) = \sum f(c)\ depth(c)$

trong đó $f(c)$ là tần số và $depth(c)$ là độ dài đường đi từ gốc đến lá tương ứng với ký tự *c*.

2.4. **Thuật toán tham lam xây dựng cây Huffman của dãy N ký tự**.

Bước 1: Tạo N cây, mỗi cây chỉ có một nút gốc, mỗi nút gốc chỉ chứa một kí tự và trọng số (tần suất của ký tự đó trong dãy N ký tự) với N = số ký tự.

Bước 2: Lặp lại thao tác sau cho đến khi chỉ còn 1 cây duy nhất:

- *Ghép 2 cây con có trọng số gốc nhỏ nhất thành 1 nút cha, có tổng ký tự, tổng trọng số trọng số của 2 nút con.*
- *Xóa các cây đã duyệt.*
- *Điều chỉnh lại cây nếu vi phạm tính chất*

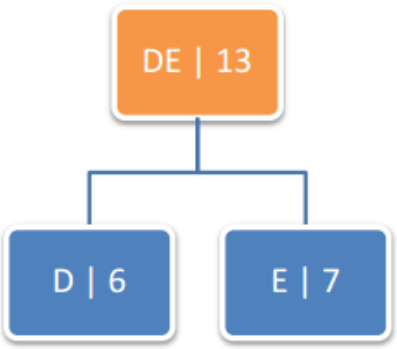
Ví dụ : Xây dựng cây Huffman để xác định mã huffman cho dãy ký tự cần nén sau :

F = “ABABBCBBDEEEABABBAEEDDCCABBBCDEEDCBCCCCDBBBCAAA” với N=47

Bảng tần suất xuất hiện

Ký tự	Tần suất
A	9
B	15
C	10
D	6
E	7

Ký tự	Tần suất
B	15
C	10
A	9
E	**7**
D	**6**

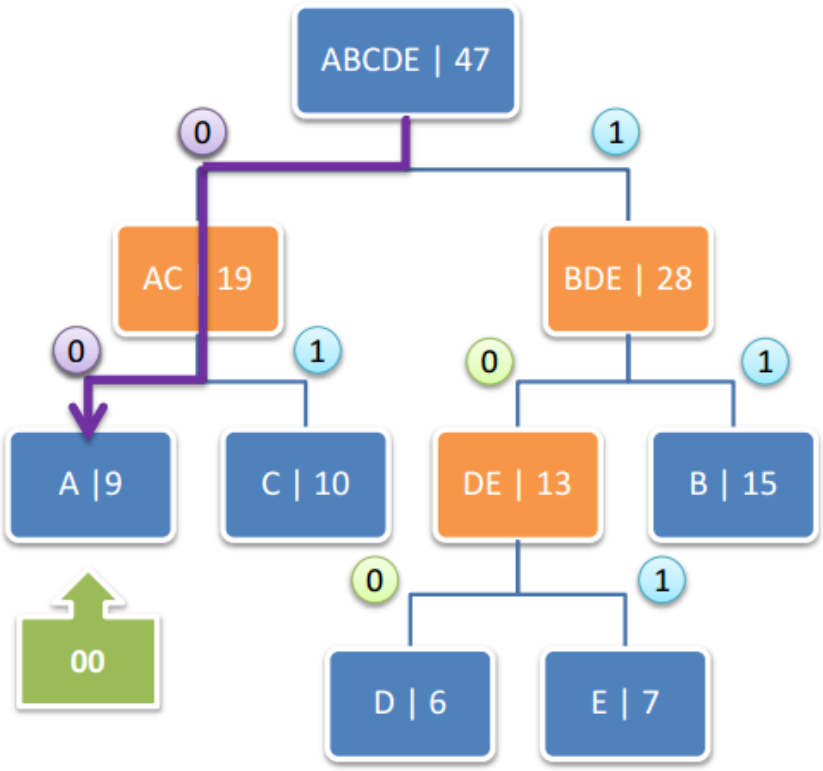


Ký tự	Tần suất
B	15
DE	13
C	**10**
A	**9**

Ký tự	Tần suất
AC	19
B	**15**
DE	**13**

Ký tự	Tần suất
BDE	28
AC	**19**

Ký tự	Tần suất
ABCDE	47



Bảng mã Huffman

Ký tự	Mã Huffman
A	00
B	11
C	01
D	100
E	101

Ký tự	Mã Huffman
A	00
B	11
C	01
D	100
E	101

Chuỗi ký tự cần nén

F = "ABABBCBBDEEEABABBAEEDDCCABBBCDEEDCBCCCCDBBBCAAA"

Chuỗi đã được nén

F_{Output} =
"00110011110111111001011011010011001111001011011001000101001111110110010110110001110101010110011111101000000"

Tiết kiệm: 8*47 - (2*9 + 2*15 + 2*10 + 3*6 + 3*7) = 376 - 107 = 269 bit
Tỷ lệ nén: (1 - 107/376)*100 = 72.54 %

2.5. Thuật toán giải nén

Bước 1: Xây dựng lại cây Huffman từ thông tin giải mã đã lưu.

Bước 2: Duyệt file, đọc lần lượt từng bit trong file nén và duyệt cây.

Bước 3: Xuất ký tự tương ứng khi duyệt hết nút lá

Bước 4: Thực hiện Bước 2, Bước 3 cho đến khi duyệt hết file.

Bước 5: Xuất file đã giải nén.

Nhận xét

- Ưu điểm :
  - Hệ số nén tương đối cao.
  - Phương pháp thực hiện tương đối đơn giản.
  - Đòi hỏi ít bộ nhớ
- Nhược điểm :
  - Mất 2 lần duyệt file khi nén.
  - Phải lưu trữ thông tin giải mã vào file nén.
  - Phải xây dựng lại cây Huffman khi giải nén.

PHẦN 3. MỘT SỐ BÀI TOÁN GIẢI QUYẾT BẰNG PHƯƠNG PHÁP QUI HOẠCH ĐỘNG

1. Giới thiệu

Khi tìm hiểu kỹ thuật chia để trị, bài toán thường bị dẫn đến một phương án có giả thuật đệ quy. Trong số các giải thuật đó, có thể có một số giải thuật có độ phức tạp thời gian mũ. Tuy nhiên, thường chỉ có một số đa thức các bài toán con, điều đó có nghĩa là chúng ta đã phải giải một số bài toán con nào đó nhiều lần. Để tránh việc giải dư thừa một số bài toán con, chúng ta tạo ra một bảng để lưu trữ kết quả của các bài toán con và khi cần chúng ta sẽ sử dụng kết quả đã được lưu trong bảng mà không cần phải giải lại bài toán đó. Lấp đầy bảng kết quả các bài toán con theo một quy luật nào đó để nhận được kết quả của bài toán ban đầu (cũng đã được lưu trong một số ô nào đó của bảng) được gọi là quy hoạch động.

Tóm tắt giải thuật quy hoạch động :

- *Bước 1 : Tạo bảng bằng cách:*

+ *Bước 1.1 : Gán giá trị cho một số ô nào đó.*

+ *Bước 1.2 : Gán trị cho các ô khác nhờ vào giá trị của các ô trước đó.*

- *Bước 2 : Tra bảng và xác định kết quả của bài toán ban đầu.*

Ưu điểm của kỹ thuật quy hoạch động là chương trình thực hiện nhanh do không phải tốn thời gian giải lại một bài toán con đã được giải. Kĩ thuật quy hoạch động có thể vận dụng để giải các bài toán tối ưu, các bài toán có công thức truy hồi.

Phương pháp quy hoạch động sẽ không đem lại hiệu quả trong các trường hợp sau:

- Không tìm được công thức truy hồi.
- Số lượng các bài toán con cần giải quyết và lưu giữ kết quả là rất lớn.
- Sự kết hợp lời giải của các bài toán con chưa chắc cho ta lời giải của bài toán ban đầu.

Sau đây là một số bài toán điển hình có thể giải bằng kỹ thuật quy hoạch động

2. Một số bài toán minh họa

2.1. **Bài toán tính số tổ hợ**p

Một bài toán khá quen thuộc là tính số tổ hợp chập k của n theo công thức truy hồi:

$$C_n^k = \begin{cases} 1 \text{ nếu } k = 0 \text{ hoặc } k = n \\ C_{n-1}^{k-1} + C_{n-1}^{k} \text{ nếu } 0 < k < n \end{cases}$$

Dựa vào công thức truy hồi trên, chúng ta có giải thuật đệ quy cho bài toán như sau :

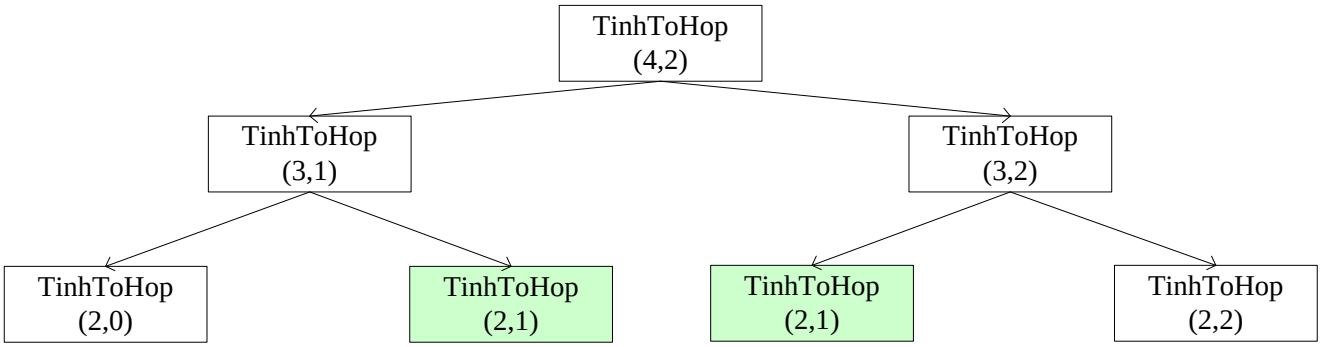
```
int TinhToHop (int n, int k)
{
        if(k==0 || k==n)            return 1;
        else
            return TinhToHop (n-1,k-1) + TinhToHop (n-1,k);
}
```

Gọi T(n) là thời gian để tính số tổ hợp chập k của n, thì ta có phương trình đệ quy:

T(1) = C1 và T(n) = 2T(n-1) + C2.

Giải phương trình này ta được T(n) = O(2^n), như vậy là một giải thuật thời gian mũ, trong khi chỉ có một đa thức các bài toán con. Điều đó chứng tỏ rằng có những bài toán con được giải nhiều lần.

Chẳng hạn để tính TinhToHop (4,2) ta phải tính TinhToHop (3,1) và TinhToHop (3,2). Để tính TinhToHop (3,1) ta phải tính TinhToHop (2,0) và TinhToHop (2,1). Để tính TinhToHop (3,2) ta phải tính TinhToHop (2,1) và TinhToHop (2,2). Như vậy để tính TinhToHop (4,2) ta phải tính TinhToHop (2,1) hai lần.



Hình 1. Sơ đồ thực hiện tinhToHop(4,2)

Áp dụng kĩ thuật quy hoạch động để khắc phục tình trạng trên, ta xây dựng một bảng gồm n+1 dòng (từ 0 đến n) và n+1 cột (từ 0 đến n) và điền giá trị cho O(i,j) theo quy tắc sau: (Quy tắc tam giác Pascal):

- C(0,0) = 1;
- C(i,0) =1;
- C(i,i) = 1 với 0 < i ≤ n;
- C(i,j) =C(i-1,j-1) + C(i-1,j) với 0 < j < i ≤ n.

Với O(n,k) là tinhToHop (n,k)

Cụ thể với n=4 ta có bảng như sau :

i \ j	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

Giải thuật quy hoạch động của bài toán tính tổ hợp :

```
int tinhToHop_DP (int n, int k)
{
        int C[n+1][n+1];
/*1*/   C[0][0] = 1;
/*2*/   for (int i = 1; i <= n; i++)
        {
/*3*/     C[i][0] = 1;
/*4*/     C[i][i] = 1;
/*5*/     for (int j = 1; j < i; j++)
          C[i][j]=C[i-1][j-1]+C[i-1][j];
        }
/*6*/   return C[n][k];
}
```

- Vòng lặp /*5*/ thực hiện i-1 lần, mỗi lần O(1).
- Vòng lặp /*2*/ có i chạy từ 1 đến n, nên nếu gọi T(n) là thời gian thực hiện giải thuật thì ta có: $T(n) = \sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2} = O(n^2)$.

Như vậy, qua việc xác định độ phức tạp, ta thấy rõ giải thuật quy hoạch động hiệu quả hơn nhiều so với giải thuật đệ quy ($n^2 < 2^n$). Tuy nhiên việc dùng bảng (mảng 2 chiều) như trên còn lãng phí ô nhớ, vì có gần 1 nửa số ô nhớ không cần dùng đến. Do đó chúng ta sẽ cải tiến thêm một bước bằng cách sử dụng véc-tơ (mảng 1 chiều) để lưu kết quả trung gian. Cách cải tiến như sau :

Dùng một véc-tơ V có n+1 phần tử từ V[0] đến V[n]. Véc-tơ V sẽ lưu trữ các giá trị tương ứng với dòng i trong tam giác Pascal ở trên. Trong đó V[j] lưu trữ giá trị số tổ hợp chập j của i (C_i^j) (j = 0 đến i). Dĩ nhiên do chỉ có một véctơ V mà phải lưu trữ nhiều dòng i do đó tại mỗi bước, V chỉ lưu trữ được một dòng và ở bước cuối cùng, V lưu trữ các giá trị ứng với i = n, trong đó V[k] chính là C_n^k.

Ban đầu, ứng với i =1, ta cho V[0] = 1 và V[1] = 1. Tức là $C_1^0 = 1$ *và* $C_1^1 = 1$. Với các giá trị i từ 2 đến n, ta thực hiện như sau :

V[0] = 1, tức là $C_i^0 = 1$. Tuy nhiên giá trị V[0] = 1 đã được gán ở trên nên không cần gán lại.

Với j từ 1 đến i-1, ta vẫn áp dụng công thức $C_i^j = C_{i-1}^{j-1} + C_{i-1}^j$, nghĩa là để tính các giá trị dòng i, ta phải dựa vào dòng i-1. Nhưng do chỉ có 1 véc-tơ V và lúc này nó lưu giá trị dòng i, còn dòng i-1 không còn. Để khắc phục điều này ta dùng thêm 2 biến trung gian p1 và p2. Trong đó p1 = C_{i-1}^{j-1} và p2 = C_{i-1}^j. Khởi đầu p1 = V[0] (=C_{i-1}^0) và p2=V[j] (=C_{i-1}^j). Sau đó V[j] lưu giá trị C_i^j sẽ được gán

bởi p1+p2. Tiếp theo p1 được gán bởi p2, nghĩa là khi j tăng lên 1 đơn vị thành j+1 thì p1 là C_{i-1}^{j} và nó được dùng để tính C_i^{j+1}.

Cuối cùng với j=i ta gán V[i] = 1 (=C_i^i=1).

Giải thuật cải tiến như sau :

```
int tinhToHop_DP_CaiTien (int n, int k)
{
        int V[n+1];
        int p1, p2;
/*1*/   V[0] = 1;
/*2*/   V[1] = 1;
/*3*/   for (int i = 2; i <= n; i++)
        {
/*4*/       p1 = V[0];
/*5*/       for (int j = 1; j < i; j++)
            {
/*6*/           p2 = V[j];
/*7*/           V[j]= p1+p2;
/*8*/           p1= p2;
            }
/*9*/       V[i] = 1;
}
/*10*/  return V[k];
}
```

Độ phức tạp của giải thuật vẫn là $O(n^2)$.

2.2. Bài toán ba lô.

Xét lại bài toán cái ba lô trong phần kỹ thuật tham lam như sau:

Cho một cái ba lô có thể đựng một trọng lượng W và n loại đồ vật, mỗi đồ vật i có một trọng lượng g_i và một giá trị v_i. Tất cả các loại đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá W và tổng giá trị là lớn nhất.

Ta sử dụng kỹ thuật quy hoạch động để giải với các số liệu được cho đều là số nguyên.

Giả sử:

- X[k,V]: là số đồ vật k được chọn.

- F[k,V]: là tổng giá trị của k đồ vật đã được chọn.
- V là trọng lượng còn lại của ba lô, k=1..n, V=1..W.

Trong trường hợp đơn giản nhất, khi chỉ có một đồ vật, ta tính được X[1,V] và F[1,V] với mọi V từ 1 đến W là: $X[1,V] = V/g_1$ và $F[1,V] = X[1,V] * v_1$.

Giả sử ta đã tính được F[k-1,V], khi có thêm đồ vật thứ k, ta sẽ tính được F[k,V], với mọi V từ 0 đến W. Cách tính như sau:

- Nếu ta chọn x_k đồ vật loại k, thì trọng lượng còn lại của ba lô dành cho k-1 đồ vật từ 1 đến k-1 là $U = V-x_k*g_k$.
- Tổng giá trị của k loại đồ vật đã được chọn $F[k,V] = F[k-1,U] + x_k*v_k$, với x_k thay đổi từ 0 đến y_k ($y_k = V/g_k$) và ta sẽ chọn x_k sao cho F[k,V] lớn nhất.

Ta có công thức truy hồi như sau:

- $X[1,V] = V / g_1$ và $F[1,V] = X[1,V] * v_1$.
- $F[k,V] = Max (F[k-1,V-x_k*g_k] + x_k*v_k)$ với x_k chạy từ 0 đến V/g_k.
- Sau khi xác định được F[k,V] thì X[k,V] là x_k ứng với giá trị F[k,V] được chọn trong công thức trên.

Để lưu các giá trị trung gian trong quá trình tính F[k,V] theo công thức truy hồi trên, ta sử dụng một bảng gồm n dòng từ 1 đến n, dòng thứ k ứng với đồ vật loại k và W+1 cột từ 0 đến W, cột thứ V ứng với trọng lượng V. Mỗi cột V bao gồm hai cột nhỏ, cột bên trái lưu F[k,V], cột bên phải lưu X[k,V]. Trong lập trình ta sẽ tổ chức hai bảng tách rời là F và X.

Ví dụ: Xét bài toán cái ba lô với trọng lượng W=9, và 5 loại đồ vật được cho trong bảng sau:

Đồ vật	Trọng lượng (g_i)	Giá trị (v_i)
1	3	4
2	4	5
3	5	6
4	2	3
5	1	1

Ta có bảng F[k,V] và X[k,V] như sau, trong đó mỗi cột V có hai cột con, cột bên trái ghi F[k,V] và cột bên phải ghi X[k,V].

k \ V	0		1		2		3		4		5		6		7		8		9	
1	0	0	0	0	0	0	4	1	4	1	4	1	8	2	8	2	8	2	12	3
2	0	0	0	0	0	0	4	0	5	1	5	1	8	0	9	1	10	2	12	0
3	0	0	0	0	0	0	4	0	5	0	6	1	8	0	9	0	10	0	12	0
4	0	0	0	0	3	1	4	0	6	2	7	1	9	3	10	2	12	4	13	3
5	0	0	1	1	3	0	4	0	6	0	7	0	9	0	10	0	12	0	13	0

Trong bảng trên, việc điền giá trị cho dòng 1 rất đơn giản bằng cách sử dụng công thức: X[1,V] = V/g_1 và F[1,V]=X[1,V]*v_1.

Từ dòng 2 đến dòng 5, phải sử dụng công thức truy hồi:

F[k,V] = Max (F[k-1,V-x_k*g_k] + x_k *v_k) với x_k chạy từ 0 đến V/g_k.

Chẳng hạn để tính F[2,7] ta có x_k chạy từ 0→ V/g_k, trong trường hợp này là x_k chạy từ 0→ 7/4. Vậy x_k có hay giá trị 0 và 1.

Khi đó F[2,7] = Max (F[2-1, 7-0*4] + 0*5, F[2-1,7-1 *4] + 1*5).

= Max(F[1,7], F[1,3] + 5) = Max(8, 4+5) = 9.

F[2,7] = 9 ứng với x_k = 1, do đó X[2,7] =1.

Vấn đề bây giờ là cần phải tra trong bảng trên để xác định phương án.

Ban đầu trong lượng còn lại của ba lô V=W.

Xét các đồ vật từ n đến 1, với mỗi đồ vật k, ứng với trọng lượng còn lại V của ba lô nếu X[k,V] > 0 thì chọn X[k,V] đồ vật loại k. Tính lại V = V - X[k,V] * g_k.

Ví dụ, trong bảng trên, ta sẽ xét các đồ vật từ 5 đến 1. Khởi đầu V = W = 9.

Với k = 5, vì X[5,9] = 0 nên ta không chọn đồ vật loại 5.

Với k = 4, vì X[4,9] = 3 nên ta chọn 3 đồ vật loại 4. Tính lại V = 9-3*2 = 3.

Với k = 3, vì X[3,3] = 0 nên ta không chọn đồ vật loại 3.

Với k = 2, vì X[2,3] = 0 nên ta không chọn đồ vật loại 2.

Với k = 1, vì X[1,3] = 1 nên ta chọn 1 đồ vật loại 1. Tính lại V = 3 – 1 * 3 = 0.

Vậy tổng trọng lương các vật được chọn là 3 * 2 + 1 * 3 = 9. Tổng giá trị các vật được chọn là 3 * 3 + 1 * 4 = 13.

Giải thuật theo kĩ thuật quy hoạch động như sau:

Phần tổ chức dữ liệu:

- Mỗi đồ vật được biểu diễn bởi một cấu trúc (struct) gồm các thông tin:
  + Ten: Lưu trữ tên đồ vật.
  + Trong_luong: Lưu trữ trọng lượng của đồ vật.
  + Gia_tri: Lưu trữ giá trị của đồ vật
  + Phuong_an: Lưu trữ số lượng đồ vật được chọn theo phương án.
- Danh sách các đồ vật được biểu diễn bởi một mảng các đồ vật.
- Bảng được biểu diễn bởi một mảng hai chiều các số nguyên để lưu trữ các giá trị F[k,v] và X[k,v].

Phần cài đặt bằng C/C++:

```
const int MAX =10;
struct Do_vat{
    char Ten[20];
    int Trong_luong,
     int Gia_tri;
```

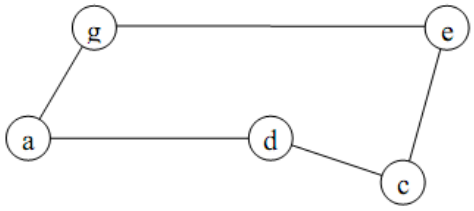
```
    int Phuong_an;
};
typedef Do_vat Danh_sach_vat [MAX]; // định nghĩa mảng 1 chiều kiểu Do_vat bằng tên
                                    //Danh_sach_vat
typedef int Bang[10][100]           // định nghĩa mảng 2 chiều kiểu int bằng tên
                                    //Bang
void Tao_Bang (Danh_sach_vat ds_vat, int n, int W, Bang F, Bang X)
 {
    int xk, yk, k;
    int FMax, XMax, v;
    /*Lấp đầy hàng đầu tiên của 2 bảng*/
    for (v= 0; v <= W; v++)
     {
       X[1][v] = v/ds_vat[1].trong_luong;
        F[1][v] = X[1] [v] * ds_vat[1].gia_tri;
    }
    /*Lấp đầy các hàng còn lại*/
    for (k = 2; k <= n; k++)
     {
            X[k][0] = 0;
            F[1][0] = 0;
     for ( v=1; v <= W; v++)
     {
            Fmax = F[k-1][v] ;
            XMax = 0;
            yk = v/ds_vat[k].trong_luong;
            for (xk=1; xk <= yk; xk++)
               if ( F[k-1][v-xk * ds_vat[k].trong_luong] + xk * ds_vat[k].gia_tri >FMax)
            {
                    FMax=F[k-1][v-k*ds_vat[k].trong_luong] + xk*ds_vat[k].gia_tri;
                    XMax= xk;
            }
            F[k][v] = FMax;
            X[k][v] = XMax;
      }
     }
}
//-------------------------------------------------------------------------------------------
// hàm tra bảng
void Tra_Bang(Danh_sach_vat ds_vat, int n, int W Bang F, Bang X)
{
    int k, v;
    v = W;
    for (k = n; k >=1; k--)
            if (X[k][v] > 0)
```

```
        {
            ds_vat[k].Phuong_an := X[k][v];
            v = v - X[k][ v] * ds_vat[k].trong_luong;
        }
}
```

2.3. Bài toán đường đi của người giao hàng

Xét lại bài toán tìm đường đi của người giao hàng ở mục 5.2.3.1, chúng ta áp dụng kỹ thuật quy hoạch động để giải như sau.

Đặt S = {x_1, x_2, …, x_k} là tập hợp con các cạnh của đồ thị G = (V, E). Ta nói rằng một đường đi P từ v đến w phủ lên S nếu P = {v, x_1, x_2, …, x_k, w}, trong đó x_i có thể xuất hiện ở một thứ tự bất kì, nhưng chỉ xuất hiện duy nhất một lần. Ví dụ đường cho trong hình sau, đi từ a đến a, phủ lên {c, d, e, g}.



Ta định nghĩa d(v, w, S) là tổng độ dài của đường đi ngắn nhất từ v đến w, phủ lên S. Nếu không có một đường đi như vậy thì đặt d(v, w, S) = ∞. Một chu trình Hamilton nhỏ nhất C_{min} của G phải có tổng độ dài là c(Cmin) = d(v,v, V - {v}). Trong đó v là một đỉnh nào đó của V. Ta xác định C_{min} như sau:

Nếu |V| = 1 (G chỉ có một đỉnh) thì c(C_{min}) = 0, ngược lại ta có công thức đệ qui để tính d(v, w, S) là:

d(v,w,{}) = c(v,w)

d(v,w,S) = min {c(v, x) + d(x, w, S – {x}}, lấy $\forall x \in S$

Trong đó c(v,w) là độ dài của cạnh nối 2 đỉnh v và w nếu nó tồn tại hoặc là ∞ nếu ngược lại. Dòng thứ 2 trong công thức đệ qui ứng với tập $S \neq \emptyset$, nó chỉ ra rằng đường đi ngắn nhất từ v đến w phủ lên S, trước hết phải đi đến một đỉnh x nào đó trong S và sau đó là đường đi ngắn nhất từ x đến w, phủ lên tập S-{x}.

Bằng cách lưu trữ các đỉnh x trong công thức đệ qui nói trên, chúng ta sẽ thu được một chu trình Hamilton tối tiểu.

TÀI LIỆU THAM KHẢO

1. **https://en.wikipedia.org/wiki/Greedy_algorithm**
2. **Slide** LECTURE 15 - Greedy Algorithms II
3. Introduction to algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
4. Bài giảng Phân tích thiết kế và đánh giá thuật toán – Khoa CNTT, Trường ĐH Hàng Hải, 2010.